2

REPRESENTING CONCEPTUAL STRUCTURES IN A NEURAL NETWORK

Technical Report AIP-12

David S. Touretzky

Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

The Artificial Intelligence and Psychology Project

Departments of
Computer Science and Psychology
Carnegie Mellon University

Learning Research and Development Center
University of Pittsburgh

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release;
Distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
AIP - 12

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Carnegie-Mellon University

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION
Computer Sciences Division
Office of Naval Research (Code 1133)

6c. ADDRESS *(City, State, and ZIP Code)*
Department of Psychology
Pittsburgh, Pennsylvania 15213

7b. ADDRESS *(City, State, and ZIP Code)*
800 N. Quincy Street
Arlington, Virginia 22217-5000

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
Same as Monitoring Organization

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
N00014-86-K-0678

8c. ADDRESS *(City, State, and ZIP Code)*

10. SOURCE OF FUNDING NUMBERS p400005ub201/7-4-86

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO
N/A	N/A	N/A	N/A

11. TITLE *(Include Security Classification)*
Representing Conceptual Structures in a Neural Network

12. PERSONAL AUTHOR(S)
D.S. Touretzky

13a. TYPE OF REPORT
Technical

13b. TIME COVERED
FROM 86Sept15 TO 91Sept14

14. DATE OF REPORT *(Year, Month, Day)*
87 September 29

15. PAGE COUNT
8

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Artificial Intelligence, Machine learning, connectionism, short-term memory, distributed representation, frames.

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

DUCS is an experimental multi-level architecture that provides a distributed representation for frame-like concept structures. The ultimate goal of this research is to develop a powerful short term memory that can construct and manipulate concepts rapidly. Such a memory could play an important role in neural network models of higher level cognitive phenomena such as language understanding or problem solving. Keywords.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
□ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT □ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION

22a. NAME OF RESPONSIBLE INDIVIDUAL
Dr. Alan L. Meyrowitz

22b. TELEPHONE *(Include Area Code)*
(202) 696-4302

22c. OFFICE SYMBOL
N00014

83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Proceedings of the IEEE First Annual International Conference on Neural Networks.
June 21-24, 1987. San Diego, California.

Representing Conceptual Structures in a Neural Network

David S. Touretzky
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

1. Introduction

DUCS is an experimental multi-level architecture that provides a distributed representation for frame-like concept structures. (The name stands for Dynamically Updatable Conceptual Structures.) The ultimate goal of this research is to develop a powerful short term memory that can construct and manipulate concepts rapidly. Such a memory could play an important role in neural network models of higher level cognitive phenomena such as language understanding or problem solving. As an example, consider how one might represent the knowledge that "Down by the hen house, John threw a rock at the fox." With a case frame representation[1], this sentence could be represented by the following structure:

AGENT:	JOHN
VERB:	THROW
OBJECT:	ROCK
DESTINATION:	FOX
LOCATION:	HEN HOUSE

In this paper I show how such structures can be realized in DUCS as distributed activity patterns over a space of neuron-like computing elements. DUCS can retrieve a slot filler from its concept buffer given the slot name; it can also retrieve entire concepts from its concept memory given some name/filler pairs as cues. In order to be useful as a short term concept memory, DUCS was designed to have the following properties:

- Concepts can be created, modified, or deleted simply by changing an activity pattern; it is not necessary to go through a series of incremental weight changes and resettlings.
- Concepts can contain any number of slots. Accuracy of retrieval degrades gradually as the number of slots stored exceeds the capacity of the network.
- Individual slots of a concept can be accessed by associative retrieval. If the pattern for a slot name doesn't exactly match what was previously stored, *e.g.*, if we try to access an elephant's TRUNK slot by referring to his NOSE, assuming the two slot names have similar but not identical microfeatures the retrieval will succeed and simultaneously the query will be corrected to read TRUNK.
- More than one slot of a concept can be accessed simultaneously, *e.g.*, it is possible to access the AGENT and the OBJECT slots of an action in parallel.
- Concepts can be associatively retrieved by combining several name/filler pairs into one cue. For example, if John did things in various places and if several things happened at the hen house, to

[1] McClelland and Kawamoto (1986) have also explored connectionist case frame representations. Their model performs lexical disambiguation and case role assignment for single sentences using four fixed cases.

recall what action John took at the hen house one combines the cues AGENT: JOHN and LOCATION: HEN HOUSE.

- Concepts that have been accessed recently remain in the concept memory, while those that have not been accessed fade out.

Perhaps the simplest way to implement concept structures (not the way chosen by DUCS) is as a collection of three-part bit vectors of form (**concept-tag slot-name slot-filler**). The tags could be meaningful patterns or randomly-generated ones; their purpose is to group together slots belonging to the same concept. A collection of these tripartite patterns could be stored in an associative memory, and given a concept tag and slot name it would be easy to retrieve the corresponding filler. However, this approach lacks several other properties important for a short term concept memory.

One issue is the need to access multiple slots in parallel. In a conventional associative memory, such as Hopfield's or Willshaw's models (Hopfield, 1982; Willshaw, 1981), information resides in the weights between units. In order to access several slots in parallel one must make several copies of the associative network and keep the weights identical in each copy. An alternative is to represent concepts as activity patterns rather than as weights, with the activity pattern transmitted from a central buffer to a number of input/output groups for retrieving different components simultaneously. DUCS uses this technique.

A second issue is the ability to associatively retrieve a particular concept by combining multiple name/filler pairs to form a cue. This also appears to require an activity-based representation for slots. One might attempt to concatenate all the slots of a concept to form one huge vector in order to effect this sort of retrieval, but since concepts contain a varying and unbounded number of slots, and the space of possible slot names is quite large, that approach would not be viable.

Finally, since the available biological evidence is that activity patterns can be modified more quickly than weights, it makes sense to explore activity-based representations when designing a short term working memory. The solution presented here is one of the key features of DUCS: it has error correcting and associative retrieval properties similar to a classical associative memory, but it is organized on different principles.

2. Overview of the Architecture

At the lowest level of DUCS, individual slots are created by setting up an activity pattern in one of several slot name groups and its associated slot filler group; see Figure 1. The name and filler patterns then jointly cause a pattern to appear in the selector group. Selector groups function as a combination pullout network (Mozer, 1984; Touretzky, 1986; Touretzky & Hinton, 1986) and viewpoint mapping device (Hinton, 1981a); they do the real work of storing and retrieving things. The detailed structure of the various groups will be described in the next section.

Slots are collected into concepts in DUCS by superimposing in the concept buffer the activity patterns of all the selector groups. Conversely, slots are extracted from the concept buffer by clamping their names into slot name groups and allowing the selector and slot filler groups to settle; the corresponding filler for each slot then appears in the associated slot filler group.

The highest level of DUCS is the concept memory, which can hold multiple concepts simultaneously. New concepts enter the memory one at a time through the concept buffer. Concepts can be recalled by combining some slots in the concept buffer and using the resulting activity pattern as a cue to associatively retrieve and fill in the rest of the concept buffer pattern from concept memory.

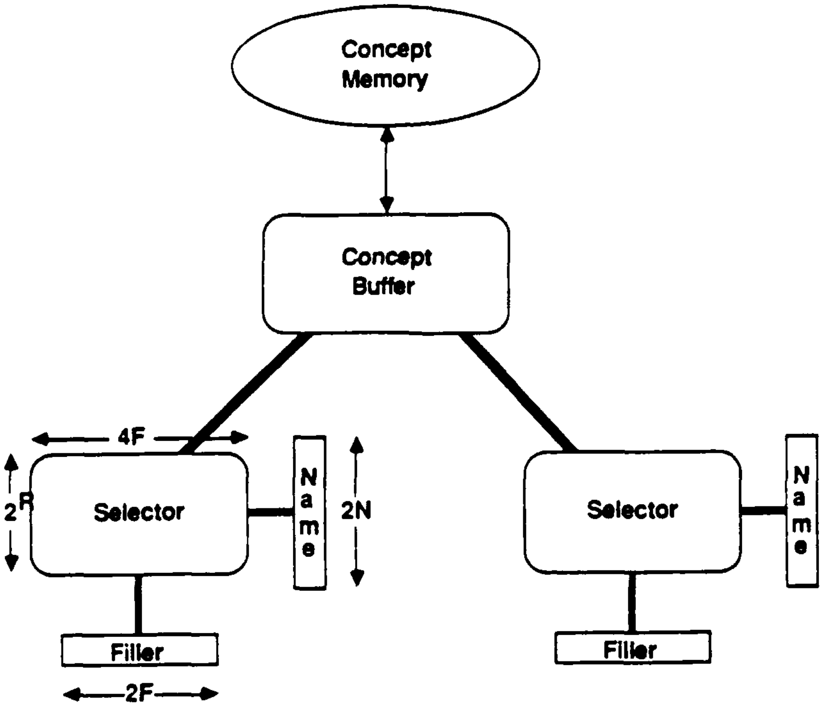

Figure 1: The DUCS Architecture.

3. A Glimpse of the Details

This section presents a concise summary of the wiring pattern of DUCS. To begin, slot names and slot fillers are N-bit and F-bit binary feature vectors, respectively, appended to their logical complements. That is, a slot name $\vec{a}$ is a $2N$-bit vector where $a_i = \bar{a}_{(i+N)}$ for $1 \leq i \leq N$, and a slot filler $\vec{v}$ is a $2F$-bit vector where $v_i = \bar{v}_{(i+F)}$ for $1 \leq i \leq F$. The selector groups and the concept buffer are both organized as $4F \times 2^R$ arrays of units, where R is a parameter between 0 and N. The arrays are $4F$ bits wide because slot fillers are stored redundantly — each bit and its complement are represented twice. Simple voting circuitry shown in Figure 2 is used to arrive at the correct slot filler vector during retrieval even when some of the concept buffer or selector units are in error. Such errors are unavoidable when using distributed representations,[2] as DUCS does, unless memory is kept very sparse, because slots will overlap in the concept buffer. The error rate for retrieval of individual slots is a function of the number of slots superimposed in the concept buffer, and also of the "breadth" of the buffer, which is a function of R. For $N = 20$ and $F = 20$, with $R = 4$ the concept buffer has a breadth of 16 units and can store three to four slots before retrieval errors occur. Increasing R makes the representation more sparse, thereby reducing the chance for slots to overlap and cause errors. The accuracy of the selector groups can also be increased by widening the concept buffer to $6F$ or $8F$ units. While the degree of sparseness will be unchanged, the extra redundancy aids error correction.

Storing filler $\vec{v}$ in the slot named $\vec{a}$ generates a $4F \times 2^R$ pattern over the selector units $\{s_{i,j}\}$. Each bit v_i is copied into one of the 2^R locations in column i of the selector array, and, independently, into one of the 2^R locations in column $i + 2F$. A unit's position j within a column is based on a randomly-chosen

[2] Hinton, McClelland, and Rumelhart (1986) describe some properties and applications of distributed representations.

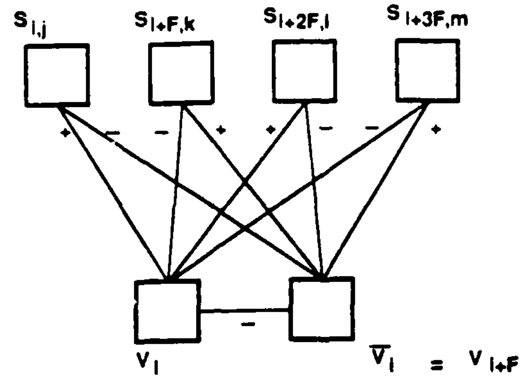

Figure 2: Error correction circuitry to expl›it the redundant storage of slot fillers. The figure shows bit v_i of a slot filler, its complement bit $\overline{v}_i$, and the corresponding selector units for four copies of the bit (two of v_i and two of $\overline{v}_i$.)

R-bit subset of $\vec{a}$; a different subset is chosen for each of the $4F$ columns. All selector groups use the same subsets for their corresponding columns. Figure 3 shows the wiring for mapping a single slot filler bit v_i into some $s_{i,j}$ and from there to the corresponding bit $b_{i,j}$ in the concept buffer. Here $R = 2$ so the the value of j is determined by two slot name bits, a_x and a_y. Note that v_i will also be copied into column $i + 2F$ in some position k, where k is determined by a (probably) different pair of slot name bits a_t and a_u. Also, $v_{(i+F)}$ which is $\overline{v}_i$ will be copied into some position in columns $i + F$ and $i + 3F$, based on two other pairs of slot name bits. This redundancy is necessary not only for error correction, but also for associative retrieval, as will be explained shortly.

Selector units function as a skeleton filter.[3] The units in each column i are organized in a mutually inhibitory winner-take-all network such that the winner will be the unit that receives excitation on every one of its R inputs from the slot name group. The winning $s_{i,j}$ functions as a mapping unit allowing activation to pass between the concept buffer unit $b_{i,j}$ and the slot filler unit v_i.

DUCS uses non-linear units with variable gain and outputs restricted to the unit interval; all connections are symmetric. This is commonly known as a Hopfield and Tank model (Hopfield & Tank, 1985). To retrieve a slot, the name is loaded into the slot name group and then it and the concept buffer are clamped, with the gain initially set very low. As the net settles, the selector units will copy the states of the chosen $b_{i,j}$'s ($1 \leq i \leq 4F$) to the slot filler group's v_i's. The variable gain feature is important for getting the retrieval to work properly, especially when the slot name does not exactly match what was stored, but the gain can rise quickly: fewer than 20 cycles are required for a retrievaL

Although increasing R increases the capacity of the concept buffer, smaller R values make it easier to retrieve a slot based on an inexact name match. For example, suppose we try to retrieve the contents of Clyde the elephant's TRUNK slot using the pattern for NOSE instead. If the two patterns differ in a single microfeature, then of the $\binom{N}{R}$ distinct R-bit subsets of the TRUNK pattern, the fraction that differ from NOSE is R/N. Assuming $R \ll N$, this fraction is small. Let slot filler unit v_i be one that happens to be mapped using the bit where the two patterns differ. Due to this difference, some selector $s_{i,k}$ rather than

[3] So called because only a skeleton subset of the units are enabled at any one time. Hinton (1981b) describes the use of skeleton filters in a connectionist implementation of a semantic network. Sejnowski (1981) presents arguments for the existence of skeleton filters in the brain.

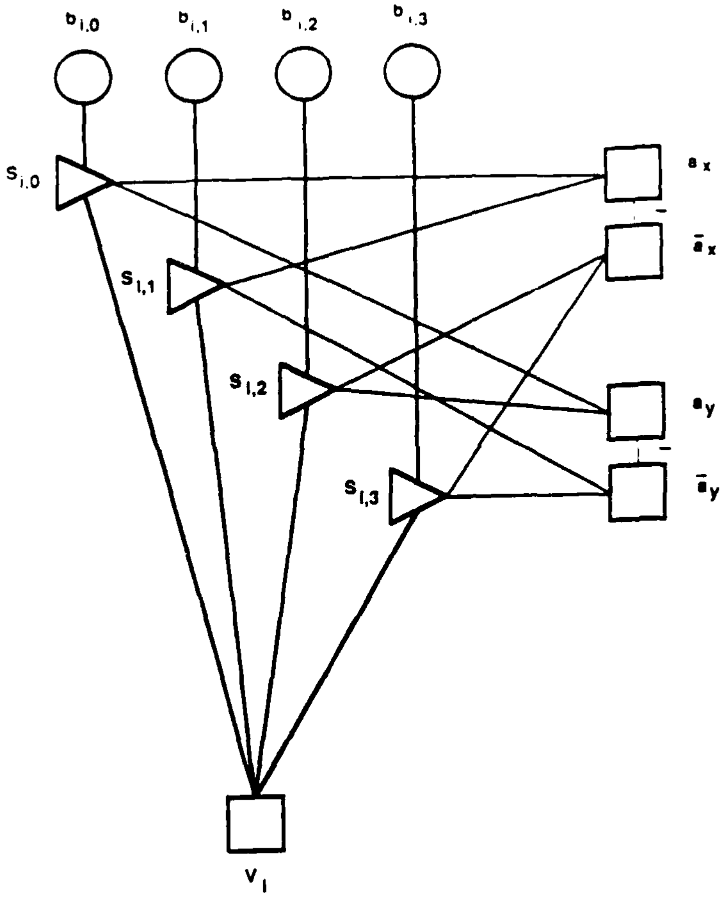

Figure 3: Hashing one copy of the slot filler bit v_i into one of 2^R concept memory bits based on a random R-bit subset of the slot name. In the figure, $R = 2$. All connections have positive weights unless shown otherwise. Mutually inhibitory connections among the 2^R selector units have been omitted for clarity.

$s_{i,j}$ will most likely be the winner in column i, and so v_i may be pressured to take on the value of $b_{i,k}$ rather than $b_{i,j}$. But if $R \ll N$, chances are very good that the redundant copies of v_i represented by selectors in columns $i + F$, $i + 2F$, and $i + 3F$ probably chose subsets that didn't include the erroneous bit, and so these three selectors will be mapped correctly and apply pressure on v_i to assume the correct value. In turn, v_i will try to make $s_{i,j}$ rather than $s_{i,k}$ the winner in column i, and due to the symmetry of connections, this will apply pressure to change the state of the erroneous slot name bit. When the slot name group is unclamped part way through the process of raising the gain during retrieval, the pattern will spontaneously change from NOSE to TRUNK. In preliminary experiments using $N = 30$, $F = 100$, and $R = 5$, the slot name did in fact correct itself during retrieval, even with two or three bit errors.

4. Design of the Weights

All weights and thresholds in DUCS are fixed. Their signs are predetermined by the model, and their magnitudes are estimated automatically when the network is initialized, based on the values of F, N, and R. A slight amount of empirical tuning is still required due to the simplicity of the current weight estimation algorithm, which does not take into account the fact that the number of connections a slot name unit makes with selector units varies with F. (The expected value is $2F \cdot 2^R \cdot R/N$, but there is some variance because the $4F$ columns of the concept buffer and selector group choose their R-sets of slot name units randomly.) Another complicating factor is the fact that the amount of input each slot name and slot filler unit receives depends on the number of slots that make up the concept, which varies from one concept to another but is generally expected to increase with R.

Slot retrieval is a constraint satisfaction problem. The formula for choosing weights is designed to balance the constraints that affect each type of unit. For example, selector unit $s_{i,j}$ should be affected equally by concept buffer unit $b_{i,j}$, slot filler units v_i and v_{i+F}, and the ensemble of R slot name units to which the selector unit connects. The selector must also be responsive to inhibitory input from the other $2^R - 1$ selectors in column i. A table of weights and their signs is given below. All connections are bidirectional, but the number of connections from each unit of type x to units of type y is not necessarily the same as the number from y to x.

			# from	# from
Connection Type	Symbol	Sign	x to y	y to x
slot filler ↔ complement	$v_i \leftrightarrow v_{i+F}$	−	1	1
slot name ↔ complement	$a_i \leftrightarrow a_{i+N}$	−	1	1
selector ↔ concept buffer	$s_{i,j} \leftrightarrow b_{i,j}$	+	1	1
selector ↔ slot name	$s_{i,j} \leftrightarrow a_x$	+	R	$\sim 2F \cdot 2^R \cdot R/N$
selector ↔ slot filler	$s_{i,j} \leftrightarrow v_i, v_{i+2F}$	+	2	2×2^R
sel. ↔ slot filler compl.	$s_{i,j} \leftrightarrow v_{i+F}, v_{i+3F}$	−	2	2×2^R
sel. ↔ competing sel.	$s_{i,j} \leftrightarrow s_{i,k}\ (j \neq k)$	−	$2^R - 1$	$2^R - 1$

5. Organization of the Concept Memory

The concept memory holds a collection of concepts, which are patterns of length $4F$ times 2^R obtained by viewing the two-dimensional concept buffer as a one-dimensional vector. To retrieve a concept, one or more cues are loaded into slot name/slot filler groups and, through the selector groups, superimposed in the concept buffer. The concept buffer then has a partial pattern which can serve as a retrieval cue for the entire concept. In the present version of DUCS the concept memory is a Willshaw-style associative memory (Willshaw, 1981), except that the one-bit weights of Willshaw's model have been replaced with simple processing units. These units can be used to detect failed retrievals (when no stored concept matches the name/filler cues supplied) and retrieval collisions (more than one concept matches the cues); more details are given in (Touretzky & Geva, 1987).

6. Discussion

DUCS is a content-independent architecture: it treats all slot name and slot filler patterns identically. Although it is convenient to assume that these patterns will be micro-feature based, so that symbols with similar meanings (like TRUNK and NOSE) have similar patterns, at present the model contains no knowledge about individual microfeatures. If it did, it could use this knowledge to further constrain the filler and slot name patterns, *e.g.*, any filler pattern containing the microfeature **human** would also have to contain the features **mammal** and **animate**. With an appropriate choice of microfeatures, slot name patterns could also be used to constrain slot filler patterns, *e.g.*, the pattern for AGENT could include a bit requiring that the filler possess the **animate** microfeature. This approach is being explored by Mark Derthick in his μKLONE system (Derthick 1987; Touretzky & Derthick, 1987). In μKLONE the microfeature patterns are specified in advance and all constraints between microfeatures are predetermined before the network is built. It might also be possible for DUCS or μKLONE to gradually learn (or at least fine tune) constraints between microfeatures by exposure to large numbers of examples.

The architecture of DUCS at the slot level bears some resemblance to the distributed ACAM (associative content addressable memory) recently described by Baum, Moody, and Wilczek (1987). The choice of a different random R-bit subset of the slot name for each column of selector units serves the same hashing function as the prime factoring approach they propose. The ACAM is a hetero-associator, and the number of weights (cf., units in DUCS) grows linearly with the size of the input and output patterns (N and F). The capacity of the memory can be increased to any desired level by adding more units. In contrast, the number of weights in a Hopfield auto-associator grows as $(N+F)^2$, assuming the patterns consist of an N-bit slot name concatenated to an F-bit filler. Since there is no intermediate layer in the Hopfield net, memory capacity is also limited; a Hopfield net can only store about $.15(N+F)$ patterns if at least 95% accuracy of recall is to be maintained.

DUCS differs from the ACAM, and most other neural networks, in that it is a two-level architecture. At the concept level, entire concepts may be stored and associatively retrieved. But each concept is a collection of named slots, and slots are themselves accessed associatively. The two-level structure is one of the key features of the DUCS model. As neural networks are applied to increasingly sophisticated knowledge representation problems, it is likely that more complex, multi-level representations will be required.

Acknowledgements

Mark Derthick's μKLONE system, still under development, was a major source of architectural ideas for DUCS. I thank Mark Derthick, Geoffrey Hinton, and Shai Geva for frequent illuminating discussions. Shai Geva implemented the concept memory portion of DUCS and also helped improve the error correction mechanism. This work was supported in part by National Science Foundation grant IST-8516330, and by the Office of Naval Research under contract number N00014-86-K-0678.

References

Baum, E. B., Moody, J., and Wilczek, F. (1987) Internal representations for associative memory. Manuscript. Institute for Theoretical Physics, University of California at Santa Barbara.

Derthick, M. A. (1987) Factual and counterfactual reasoning by constructing plausible models. *Proceedings of AAAI-87*, Seattle, WA.

Hinton, G. E. (1981a) A parallel computation that assigns canonical object-based frames of reference. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, BC, 683-685.

Hinton, G. E. (1981b) Implementing semantic networks in parallel hardware. In Hinton, G. E. and Anderson, J. A. (eds.), *Parallel Models of Associative Memory.* Hillsdale, NJ: Earlbaum.

Hinton, G. E., McClelland, J. L, and Rumelhart, D. E. (1986) Distributed representations. In D. E. Rumelhart and J. L. McClelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. Cambridge, MA: The MIT Press.

Hopfield, J. J. (1982) Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA*, vol. 79 pp. 2554-2558.

Hopfield, J. J. and Tank, D. (1985) "Neural" computation of decisions in optimization problems. *Biological Cybernetics, 52*, 141-152.

McClelland, J. L. and Kawamoto, A. H. (1986) Mechanisms of sentence processing: assigning roles to constituents. In J. L. McClelland and D. E. Rumelhart (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2. Cambridge, MA: The MIT Press.

Mozer, M. C. (1984) The perception of multiple objects: a parallel distributed processing approach. Unpublished thesis proposal, Institute for Cognitive Science, University of California at San Diego. La Jolla, CA.

Sejnowski, T. J. (1981) Skeleton filters in the brain. In Hinton, G. E. and Anderson, J. A. (eds.), *Parallel Models of Associative Memory*. Hillsdale, NJ: Earlbaum.

Touretzky, D. S. (1986) BoltzCONS: reconciling connectionism with the recursive nature of stacks and trees. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA, 522-530.

Touretzky, D. S. and Hinton, G. E. (1986) A distributed connectionist production system. Technical report CMU-CS-86-172, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA. December, 1986.

Touretzky, D. S. and Derthick, M. A. (1987) Symbol processing in connectionist networks: five properties and two architectures. *Digest of papers: COMPCON Spring 87, Thirty-Second IEEE Computer Society International Conference*, February, 1987, Cathedral Hill Hotel, San Francisco, CA, pp. 30-34.

Touretzky, D. S. and Geva S. (1987) A distributed connectionist representation for concept structures. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA.

Willshaw, D. (1981) Holography, associative memory, and inductive generalization. In Hinton, G. E. and Anderson, J. A. (eds.), *Parallel Models of Associative Memory*. Hillsdale, NJ: Earlbaum.